Design Space Exploration of Next-Generation HPC Machines

Constantino Gómez
Barcelona Supercomputing Center
Barcelona, Spain
constantino.gomez@bsc.es

Francesc Martínez
Barcelona Supercomputing Center
Barcelona, Spain
francesc.martinez@bsc.es

Adrià Armejach
Barcelona Supercomputing Center
Barcelona, Spain
adria.armejach@bsc.es

Miquel Moretó
Barcelona Supercomputing Center
Barcelona, Spain
miquel.moreto@bsc.es

Filippo Mantovani
Barcelona Supercomputing Center
Barcelona, Spain
filippo.mantovani@bsc.es

Marc Casas
Barcelona Supercomputing Center
Barcelona, Spain
marc.casas@bsc.es

***Abstract*—The landscape of High Performance Computing (HPC) system architectures keeps expanding with new technologies and increased complexity. With the goal of improving the efficiency of next-generation large HPC systems, designers require tools for analyzing and predicting the impact of new architectural features on the performance of complex scientific applications at scale. We simulate five hybrid (MPI+OpenMP) applications over 864 architectural proposals based on state-of-the-art and emerging HPC technologies, relevant both in industry and research. This paper significantly extends our previous work with MUltiscale Simulation Approach (MUSA) enabling accurate performance and power estimations of large-scale HPC systems. We reveal that several applications present critical scalability issues mostly due to the software parallelization approach. Looking at speedup and energy consumption exploring the design space (i.e., changing memory bandwidth, number of cores, and type of cores), we provide evidence-based architectural recommendations that will serve as hardware and software co-design guidelines.**

***Index Terms*—HPC, Co-design, Parallelism, OpenMP, MPI, Next-generation architectures**

I. Introduction

Trends in High Performance Computing (HPC) systems are changing. The use of commodity server-grade processors as the common choice to design these systems is moving into a more specialized landscape. Processor trends are evolving in different directions, such as, leaner core designs [1], larger core counts per socket [2], wide vector units [3], or with integrated memory like high-bandwidth memory (HBM) modules via silicon interposer technologies [4]. Consequently, the design space for next-generation HPC machines is expanding. Understanding how relevant HPC applications perform under these different design points is crucial to determine the right path when designing a new HPC system.

Hybrid programming models are commonplace nowadays, employing MPI for inter-node communication and a shared-memory programming model for node-level parallelism. Optimizing node-level performance and energy is the first step towards a balanced system; however, as the number of nodes increases with the deployment of new HPC systems, node-to-node communication costs become more relevant and need further consideration. Finding the right ratio between the number of nodes and the number of processing units per node is a primary design decision that can greatly impact application performance. Since some trends are also advocating for larger core counts, it is also important to consider the interactions with the system software at the node level (e.g. the runtime system).

In this paper, we undertake a design space exploration study that considers the most relevant design trends we are observing today in HPC systems. To perform this study, we follow a recently introduced multi-level simulation methodology (MUSA) [5]. MUSA enables fast and accurate performance estimations and takes into account inter-node communication, node-level architecture, and system software interactions. We extend MUSA in multiple ways to perform a detailed evaluation of the impact of a set of identified key design trends on relevant HPC applications.

The contributions of this paper are: *i)* We extend an end-to-end simulation methodology called MUSA. In particular, we add support for a larger set of OpenMP pragmas, multiple target architectures, a model for vectorization, performance and power modeling of emerging memory technologies, and processor power estimations. The resulting infrastructure can target a wide range of HPC applications, and provide performance and power estimations of the trends that are currently dominating in HPC. *ii)* We perform large-scale simulations comprised of 256 MPI ranks, each representing a compute node, and up to 64 cores per node adding a total of 16,384 cores. Our design space exploration methodology analyzes current HPC trends by factoring in the relevant architectural parameters, for which we derive over 800 different architectural configurations. We simulate each of the 5 selected representative HPC applications with these configurations and provide estimations for performance, power and energy to solution. *iii)* Through our extensive design space exploration, we provide hardware and software co-design recommendations for next-generation large-scale HPC systems.

Our results demonstrate that 512-bit wide FP units yield performance speedups between 20% to 75% and that they may also bring energy consumption reductions for codes that properly exploit thread- and SIMD-level parallelism. Also, cache hierarchies of around 1MB of last level cache capacity per core and moderate out-of-order capacities are a good compromise between performance and power consumption. Finally, our experiments indicate how the parallel efficiency of parallel codes is of paramount importance to avoid the waste of leakage power.

The remainder of this paper is organized as follows. In Section II, we provide background and motivation of our work. Next, we describe our work extending MUSA methodology in Section III. We introduce our experimental setup in Section IV. We perform the design space exploration of next-generation HPC systems in Section V. Finally, we discuss the related work in Section VI and share our conclusions in Section VII.

II. Co-Design of HPC Applications and Systems

A. Background

Prior work [5] introduced MUSA, a comprehensive multi-scale methodology to enable fast and accurate performance estimations of large-scale HPC machines. The methodology captures inter-node communication as well as intra-node micro-architectural and system software interactions by leveraging multi-level traces. These traces also allow for different simulation modes and execution replay to quickly extrapolate results of entire hybrid applications running on large-scale systems with tens of thousands of cores.

MUSA employs two components: (i) a tracing infrastructure that captures communication, computation and runtime system events; and (ii) a simulation infrastructure that leverages these traces for simulation at multiple levels.

Tracing: The initial step is to trace an application's execution at multiple levels. Given our targeted hybrid programming model, we start tracing each MPI process representing a rank with a single thread. This trace file contains coarse-grain information about the MPI communication phases as well as high-level computation information of the runtime system events.

In addition, MUSA requires instruction-level instrumentation for computational phases, such as the operation code, the program counter and the involved registers and memory addresses. Such detailed instrumentation is deferred to a separate native execution due to its higher overhead that might alter application behavior. Hence, when tracing in detailed mode, the timestamps taken in the initial coarse-grain trace are used to correct any deviation in the behavior of the application introduced in the detailed trace step.

This tracing methodology generates traces that allow simulations even if the characteristics of the simulated computational node (e.g., the number of cores, the memory hierarchy) or the communication network change. As a result, it is possible to perform architectural analysis of a large design space using the same set of traces, reducing trace generation time and storage requirements.

Simulation: The methodology initially identifies the different computation phases for each rank using the initial coarse-grain trace, which are independent and can be simulated in parallel. Each of these rank level computation phases is simulated with the specified number of cores, and parameters of the microarchitecture and the memory hierarchy. MUSA is able to simulate an arbitrary number of cores per rank; to accomplish this, MUSA injects runtime system API calls by using the runtime system events recorded in the trace, effectively simulating the runtime system, including scheduling and synchronization for the desired number of simulated cores.

After the computation phases have been simulated, MUSA replays the execution of the communication trace events in order to simulate the communication network and generate the final output trace of the simulation. During this process, the durations of the computation phases are replaced by the results obtained in the simulations, and the communication phases are simulated using a network simulator. At the end of this process the entire simulation is complete and the output trace is generated for visualization and inspection.

B. Trends and Challenges

The design space for next-generation HPC machines is expanding. First, the trend to use commodity server processors as the common choice is changing towards processors with leaner core designs that feature different microarchitectural characteristics. For example, Cray has already deployed Isambard [6], a system with 10,000+ Armv8 cores; and now supports ARM-based processors (including the Cavium ThunderX2) across their main product line. Second, vector architectures with larger lengths than the ones employed in recent years are starting to be considered again. In this regard, Arm recently introduced the Scalable Vector Extensions (SVE) that support up to 2,048 bit vectors and per-lane predication. Third, several memory technologies are starting to appear in the HPC domain, for example: die-stacked DRAM like the one employed in Knights Landing [7], or High-Bandwidth Memory (HBM) already used in a number of GPUs.

The advent of these trends and technologies leads to a large design space for next-generation HPC machines that needs to be carefully considered. There is a clear opportunity to co-design hardware and software by mapping application requirements to the available hardware ecosystem these trends are opening. In addition, the ability to predict and fine-tune application performance for selected hardware designs that are deemed of interest is of paramount importance to system architects. Methodologies like MUSA can therefore help understand current and future scientific application performance on systems not yet available on the market and identify the best design points.

In order to be effective, we need to ensure the employed methodology captures the trends and the characteristics of the technologies under consideration, as well as main design constraints such as power consumption. In this paper we significantly extend MUSA to support modeling of key aspects to enable accurate performance estimations of large-scale HPC

systems that will make use of the technologies mentioned before. The following section describes these extensions that are later used to perform a comprehensive design-space exploration analysis of large-scale HPC systems.

III. Extending MUSA for Practical Design-space Exploration

We extend MUSA in multiple ways that not only allow us to capture the technological trends described before, but also enable exploration of a wider range of applications on different hardware architectures. The following paragraphs describe the functionalities added to MUSA for producing the results we present in this paper.

Support for OpenMP `parallel for` *constructs:* The tracing infrastructure had support to trace applications based on tasks, both based on the OmpSs programming model [8] and the tasking features introduced in OpenMP 3.0 [9]. This was a limiting factor that did not allow us to trace applications of interest that use classic parallel for constructs, restricting exploration studies to a small set of application choices, thus narrowing co-design opportunities. For this reason, we extend the tracing infrastructure to support parallel loops as well as other common directives like *omp critical*. Therefore, MUSA is able to target a wide range of applications, making design-space exploration studies more robust and with a better coverage of application characteristics.

Support for multiple architectures: The tracing infrastructure was based on the dynamic binary translation tool PIN [10]. However, PIN is x86-specific and has some pitfalls that are not easy to overcome. For example, it is closed source code and comes with its own version of *libc*, which is not *c++11* compliant. This can lead to stiff restrictions when compiling applications that need full compliance with this standard, either directly or through any external library. To overcome these issues, we port the entire tracing infrastructure to DynamoRIO [11], which allows MUSA to support both x86_64 and Armv8 binaries while also removing all the restrictions that PIN imposes with its embedded *libc*. As we did with our PIN traces, we also validate that DynamoRIO traces are accurate. When compared to PIN traces on an Intel Xeon E5-2670, the differences in terms of loads, stores, and micro-instructions remained below 0.14%, 0.27%, and 2.40% respectively for all tested applications.

Support for vectorization: We consider a simple and practical model to enable simulations using different vector lengths while reusing the same application instruction trace. When our instruction tracing infrastructure finds a vector instruction, our internal decoder breaks them into scalar instructions with a special marker, effectively obtaining a trace with only scalar instructions. During the simulation step, if a vector length of 128 bits or wider has been requested, we fuse the marked instructions in order to simulate the specified vector length. For example, if a 128-bit vector length is specified, two arithmetic instructions are fused into one, while memory operations are also fused but its size is doubled to account for memory bandwidth. This mechanism trivially works for vector lengths of the same size or narrower than the one used during the tracing step. However, we also enable simulations with wider vector lengths by applying this fusion-driven mechanism to dynamic instructions corresponding to the same static instruction of the same basic block. We require a basic block to be executed several times in a row to apply the fusion mechanism. While this simple model may overestimate the vectorization impact, it is useful to indicate the potential that current HPC codes have for performance improvements when exposed to long SIMD register sizes.

Support for emerging memory technologies: A key component that significantly impacts the performance of current and future HPC machines is the employed memory technology. To enable MUSA to accurately model a wide range of emerging memory technologies, we add interfaces into our architectural simulator to support a fast and extensible external memory simulator, Ramulator [12]. Ramulator is able to model a wide range of commercial and academic DRAM standards, including: DDR, LPDDR, HBM, and Wide-IO. In addition, for most of these standards, Ramulator is also capable of reporting power consumption by relying on DRAMPower [13] as a backend. We integrate these tools into our toolflow, providing a robust infrastructure to obtain performance and power estimations for the memory subsystem.

Support for power estimations using McPAT: We also integrate the McPAT [14] modeling framework into our toolflow to obtain power estimations of the simulated multicore. We feed McPAT with the different architectural descriptions, as well as the simulation statistics, to obtain power estimations of the different cache levels and key core hardware structures.

The sum of these new features makes MUSA a robust tool to perform exhaustive design-space exploration studies that can cover a wide range of applications, potentially on different architectures, as well as hardware trends like vectorization and emerging memory technologies.

IV. Design space exploration

This section describes the methodology we employ to carry out our design space exploration of HPC architectures.

A. Architectural Parameters

After reviewing the HPC systems landscape, we select a set of important compute node features in current and upcoming HPC architectures. These features expose relevant energy and performance trade-offs when considering different HPC workloads. We focus our exploration on six features: number of cores in a socket, out-of-order (OoO) capabilities of the core, memory technology, floating-point unit (FPU) vector width, CPU frequency and cache size.

Per each feature, we explore a set of possible values that are based on newly added Top500 systems or near-future announced systems [1], [4], [15]–[17]. Based on this information, we use 32 to 64 cores per socket with clock frequencies ranging from 1.5GHz to 3GHz. We define four types of core pipelines by modifying their OoO capabilities: a modest but floating-point capable, close to in-order, low power core, with

L3:L2-caches	Size / associativity / latency	
Label	**L3**	**L2**
32M:256KB	32MB / 16 / 68	256kB / 8 / 9
64M:512KB	64MB / 16 / 70	512kB / 16 / 11
96M:1MB	96MB / 16 / 72	1MB / 16 / 13

Core OoO Label	ROB	Issue& commit	Store buffer	#ALU/ #FPU	IRF/ FRF
low-end	40	2	20	1 / 3	30/ 50
medium	180	4	100	3 / 3	130 / 70
high	224	6	120	4 / 3	180 / 100
aggressive	300	8	150	5 / 4	210 / 120

Other param.	Values
Frequency [GHz]	1.5, 2.0, 2.5, 3.0
Vector width [bits]	128, 256, 512
Memory [DDR4-2333]	4-channel, 8-channel
Number of Cores	1, 32, 64

TABLE I: Simulation architectural parameters and values used in our design space exploration including: cache size, associativity and latency; and OoO details like Reorder buffer (ROB) and Integer/Float Register File (RF).

three floating-point units and small issue width and buffers; two server-class cores in the medium high range; and an aggressive high-end configuration with an issue width of eight instructions, large hardware buffers, and up to four floating point units. To tweak the memory hierarchy we modify the L2 (private) and L3 (shared) cache sizes as well as the number of off-chip memory channels using DDR4-2333 technology. Lastly, we also explore the impact of using 128-bit, 256-bit and 512-bit wide floating point units. Table I shows a detailed list of all the parameters and values we explore and the names (labels) we will use to refer to them. We consider each possible combination of architectural configurations, running in total 864 simulations per application.

B. HPC Applications

To perform our experiments we use five HPC applications: HYDRO [18], a simplified version of RAMSES [19] that solves compressible Euler equations of hydrodynamics using the Godunov method; SP and BT multizone NAS benchmarks [20], which implement diagonal matrix solvers; LULESH [21], which implements a discrete approximation of the hydrodynamics equations; and Specfem3D, which uses the continuous Garlerkin spectral-element method to simulate forward and adjoint seismic wave propagation on arbitrary unstructured hexahedral meshes. For each application we adjust the input sets to potentially have enough parallelism when running on 256 MPI Ranks, one per node, and 64 cores per node, 16,384 cores in total.

All applications use a hybrid programming model, either (MPI+OpenMP) or (MPI+OmpSs). OmpSs [8] and OpenMP [9] (since ver. 3.0) allow the annotation of parallel regions as tasks with input and output dependencies. During execution, OmpSs runtime system is in charge of scheduling task instances to the compute units. We compile all applications with GCC 7.1.0 and OpenMPI [22] 1.10.4. In GCC

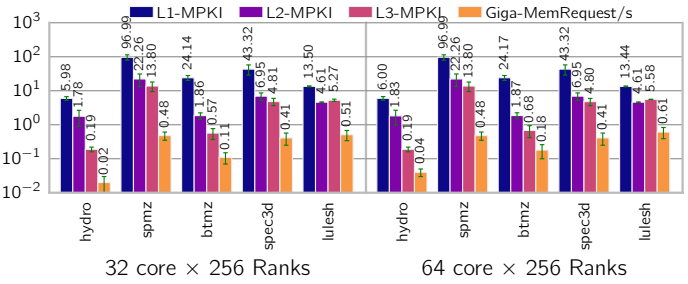


Fig. 1: Application runtime statistics: Misses Per Kilo Instruction (MPKI) of caches and Billions of requests to main memory per second.

we use the -O3 optimization flag and force -msse4.2 in order to generate binaries with Intel SSE4.2 (128-bit width) SIMD instructions.

For the purpose of workload characterization and analysis discussion, in Figure 1 we show runtime memory access statistics that we obtain in our detailed hardware simulations.

C. Tracing and Simulation Infrastructure

For our experiments, we use a toolchain that implements the MUSA methodology. It allows us to obtain traces and to perform simulations at different levels of abstraction as we describe in Section II. We obtain application traces using two lightweight tracing tools: Extrae [23] and DynamoRIO [11]. For each application we obtain two traces: a high-level trace (burst) using coarse grain instrumentation and a low-level trace using fine grain instrumented (detailed). The burst trace captures the events through the execution of the whole application. However, we only trace in detail a sample region of each application: in our applications, tracing one iteration (usually the second) of one MPI rank, is enough to capture a representative sample of the computational behavior of the application [5].

For detailed instruction-level simulations we use TaskSim [24] and Ramulator. Full application simulations are driven by Dimemas [25], which implements a high-level network model and is capable of integrating the accurate timing values that we obtain doing detailed simulations with TaskSim.

We obtain node power estimations using McPAT [14] and DRAMPower [13]. During TaskSim simulation Ramulator generates DRAM command traces that DRAMPower uses as input. Although Ramulator splits that trace into a different file for every channel and rank, DRAMPower does not support multi-rank DIMM simulation. Instead, we specify power parameters using single rank DDR4 datasheet from Micron [26]. In simulations with four channels we attach eight DIMMs for a total of 64 GB while in simulations with eight channels we attach 16 DIMMs for a total RAM of 128 GB.

Dimemas and TaskSim, have been validated using the MareNostrum III supercomputer and three MPI+OpenMP codes obtaining a $< 10\%$ relative error [5], [27]. Previous work [12] describes how Ramulator is validated. We use

the latest model adjustments in McPAT for state-of-the-art CMPs in order to improve accuracy to $< 20\%$ error [28]. DRAMPower claims to have $< 2\%$ error [29].

V. Evaluation

A. Scaling Analysis of Applications

Programming parallel applications to scale efficiently in next-generation systems where we expect to have a high number of distributed compute elements is a challenge. Understanding code parallelization scaling limits of applications is very valuable knowledge that helps to explain possible unexpected hardware interactions: for example, underusage of shared resources like memory buses and caches in memory intensive applications due to a low number of concurrent tasks in a processor. In this section, we show a scaling analysis of the five applications that are object in this study. We have been able to identify and quantify the source of parallel programming-related bottlenecks limiting scalability like MPI overheads or OpenMP task scheduling bottlenecks. We use the MUSA burst mode to obtain traces of the compute and MPI regions. These traces drive the simulation of applications in nodes with up to 64 cores. In burst mode, no details of the processor architecture are modeled and task execution time is not affected by penalties from shared cache contention or memory bandwidth exhaustion (see Section II); we call this *hardware agnostic* simulation.

In Figure 2a we simulate a single representative compute region of each application; across applications, we observe an average parallel efficiency of roughly 70% at 32 cores, dropping rapidly to 50% at 64 cores. HYDRO is the only application whose main compute region scales over a reasonable $> 75\%$ parallel efficiency in systems with 64 cores per node. We analyze with visualization tools [30] traces containing task execution and task scheduling events generated during the simulation of applications in burst mode. In them we observe that the main overall source of performance losses at 32 and 64 cores for all codes is the lack of task level parallelism. Even using relatively large input sets, the main parallel compute regions in all applications except HYDRO do not have enough fine-grain task granularity to fill all cores simultaneously. In Figure 3 we can appreciate this behavior in Specfem3D, where most tasks (colored) are scheduled only in few of the threads while the rest remain idle (middle horizontal gray area). We also find important serialized execution segments in all applications except SPMZ. Lastly, thread level load imbalance is the main issue limiting LULESH scaling potential on 64 core configurations.

To further study the additional overhead of MPI communications on top of the code parallelization issues, we integrate all compute regions together with all MPI regions. Figure 2b shows the achieved scalability without considering initial data structure allocation and final I/O operations. Same as before, we do not consider shared caches or memory bus contention effects inside the node. We simulate a network with bandwidth and latency similar to Marenostrum IV [17]. In this case, average parallel efficiency scales up to 49% and 28% when using configurations with 32 cores or 64 cores respectively. By analyzing MPI communication traces we observe that: *i)* message passing represents a minimal part of the total MPI overheads; *ii)* load imbalance, across different MPI ranks in the presence of synchronization barriers, causes a significant loss of performance in all applications except for HYDRO. In figure 4 we appreciate this behavior: colored in red the actual time spent point-to-point calls, in pink MPI_AllReduce regions causes all ranks to synchronize.

B. Hardware Exploration

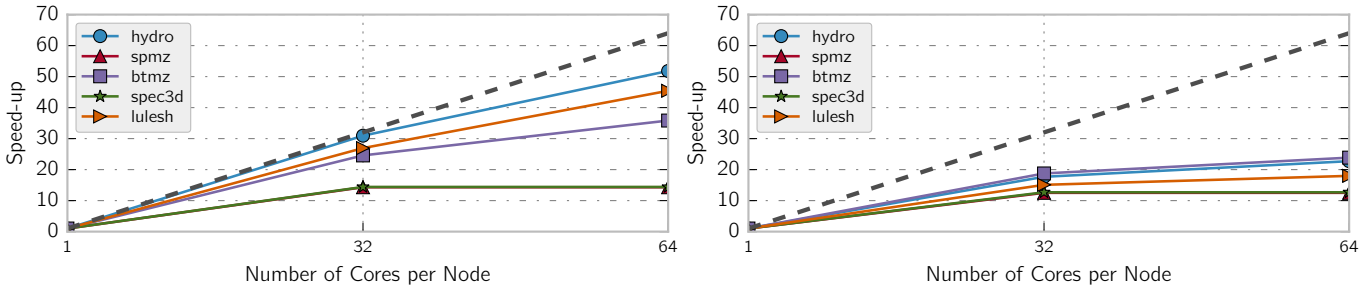
Next, we present our design space exploration based on detailed hardware simulations for six main architectural components. For each component, we discuss the results that we provide in form of performance, power dissipation and energy-to-solution figures. To quantify the performance impact of each individual component, our plots represent the average values obtained by normalizing each simulation against another simulation that shares all the other architectural parameters except the one we are quantifying.

For example, in Figure 5a we measure the impact across applications of increasing the FPU vector width. Considering that we have a total of six parameters, a simulation instance would be $\{x, y, z, s, t, 128bit\}$, where x, y, z, and t represent a specific value for each of the other architectural components parameters. Then, we normalize the execution time of each simulation with vector width = 256-bit against its baseline simulation that shares all the other architectural parameters, i.e., we would normalize $\{x, y, z, s, t, 256bit\}$ against our baseline $\{x, y, z, s, t, 128bit\}$. In this case, with a total of 864 simulations per application, we are averaging 96 samples per bar.

1) FPU vector width: Figure 5 summarizes the performance-energy trade-off when we increase the vector Floating Point (FP) registers used for SIMD operations in each core. Results for 32 and 64 core configurations are very similar. Excluding LULESH, wider 512-bit FP units yield 20% (HYDRO) to 75% (SP-MZ) application performance speed-up; 40% on average.

As we describe in Section III, the TaskSim approach to simulate SIMD instructions targets loops containing vector instructions that run for a large number of iterations. In codes with loops with a very short iteration count, like LULESH, our approach does not detect any potential for performance improvement by extending the vector size.

Measuring the *Core+L1* component in Figure 5b, we see that using 512-bit vector width translates into an average power increment across applications of 60% with respect to 128-bit units in each core. As expected, the core power consumption is relatively larger in compute-intensive applications like HYDRO and BTMZ than in memory bound counterparts. As a consequence, we appreciate a larger impact in the power consumption of applications that are compute-intensive, when increasing the vector units width. In terms of energy to solution, in all applications except LULESH, 256-bit configurations obtain 3% to 18% energy savings.



(a) Single compute region of the application.

(b) Full application parallel region.

Fig. 2: Scaling of applications using hardware agnostic simulations: a) measures a single representative compute region without MPI communications, b) measures the whole parallel region including MPI overheads.

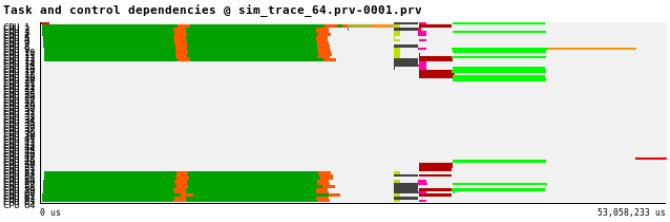


Fig. 3: Low level task parallelism is the main cause for low parallel efficiency in Specfem3D. Many CPUs remain idle (in gray) during the whole execution, while few are populated with tasks (in color). X axis represents time, the Y axis represents the thread number.

A careful analysis of the current auto-vectorization extracted from the applications under study shows that the auto-vectorization is enough to start obtaining gains from 256-bit vector units, but we lack manual vectorization or more efficient compiler auto-vectorization techniques in order to benefit from larger vectors.

We observe important timing differences when HYDRO is executed with L2 cache configurations smaller than 512KB. Configurations bottlenecked in memory reduce throughput and in consequence, diminish the stress in the individual compute units inside each core allowing them to scale better in relative terms; absolute execution time will still be lower due to the memory bottlenecks (see Fig. 6a). In a similar way, when using a smaller cache and low-end core configurations in BTMZ, the widening of FP units results in a higher relative speedup.

2) Cache sizes: Figure 6 shows how only modifying L2- and L3-cache sizes affects performance in our simulations; at 64 cores, upgrading to a cache configuration with 96MB:1MB (1.5MB:1MB per core) results in an 11% average speedup across applications.

Fitting the workload dataset in cache has a huge impact on performance. In HYDRO we obtain a $4\times$ drop in L2-cache MPKI when upgrading the L2-cache size from 256 kB to 512 kB per core, meaning that the main working set of HYDRO fits in less than 512 kB. This translates into a 21% average performance improvement for HYDRO. In the case of BTMZ and LULESH, we also obtain 9% and 12% speedup respectively. Specfem3D shows no differences across cache configurations: although it obtains locality benefits using larger caches those are minimal and it is not enough to compensate the increased latency per access to larger cache structures.

Figure 6b shows that with 64 cores, in configurations with 32MB, 64MB, and 96MB of L3-cache, the *L2+L3Cache* component represents respectively around 5%, 10% and 20% of the total power consumption. While the performance benefits when upgrading caches from 32MB:256KB to 64MB:512KB are significant in some applications; further upgrading from 64MB:512KB to 96MB:1MB at the cost of doubling the power budget of L2-/L3-cache is not justified due to smaller gains in terms of performance. Similar trends can be seen in the 32 cores configurations. Aside from Specfem3D, final energy reductions are minimal, around 5% on average for 64MB:512KB and $\sim$1% for 96MB:1MB.

Careful sizing of the L2 and L3 cache structures is necessary to minimize power consumption while achieving a good level of performance. Based on our results, we find that 1MB L3-cache and 512 kB L2-cache per core seem to offer the best trade-off. Note that we execute our applications without any manual or auto-tuning optimizations. Also, adding software optimizations, like cache blocking, to adequately fit application working sets into cache should be specially considered in systems with a high number of cores per socket to improve the energy consumption and reduce last-level cache miss overheads.

LULESH and HYDRO simulation results have a significant standard deviations of 5% and 15%, respectively. In LULESH, eight DDR4 channels compared to four DDR4 channels configurations achieve lower relative speedups when increasing L2- and L3-cache sizes. Although in absolute terms eight-channel configurations run faster, the relative improvement is lower. Also, variations in HYDRO are caused by bottlenecks in configurations with more than 2.5 GHz CPU frequency (see figure 9a).

3) Core Out-of-Order capabilities: In Table I, we define four configurations to model different levels of core Out-of-Order (OoO) capabilities. The aggressive and low-end configurations are extreme cases that differ considerably from the usual state-of-the-art type cores found in HPC systems. Be-

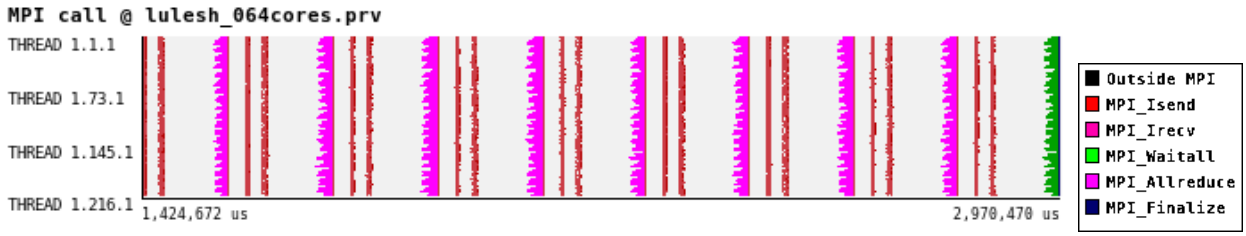


Fig. 4: Visual timeline of a trace representing MPI and compute phases. Significant unnecessary time is spent in MPI_Barriers due to load imbalance in LULESH. X axis represents time, the Y axis represents the rank number.

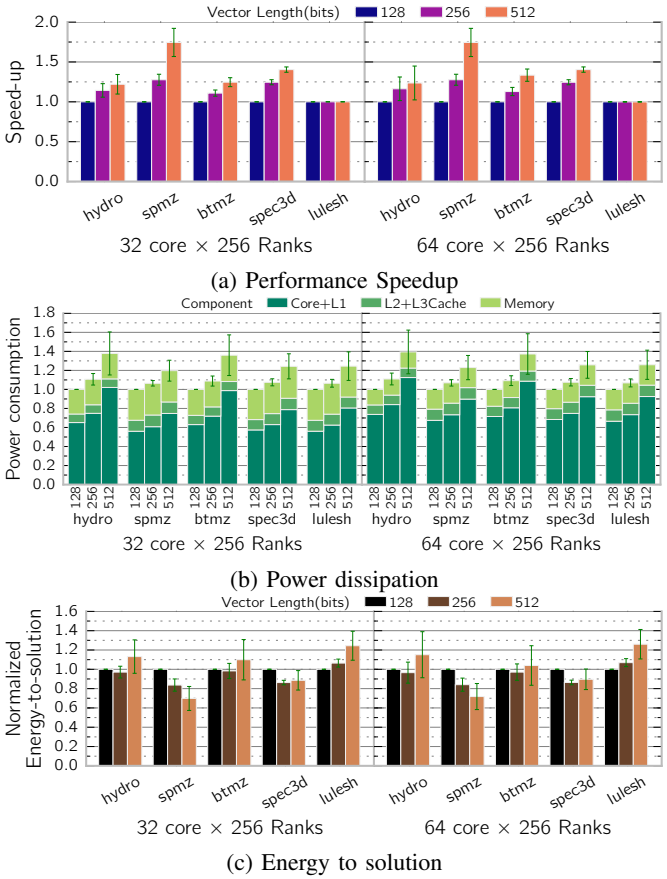


Fig. 5: Average simulation results increasing FPU width up to 512-bits. Normalized to 128-bit configurations.

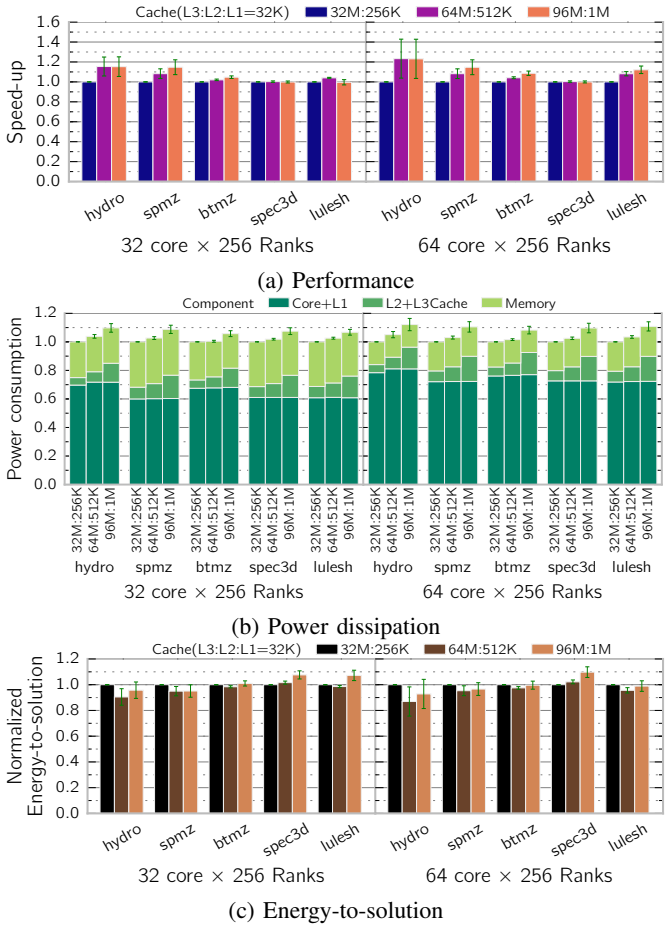


Fig. 6: Average simulation results varying L3- and L2-cache parameters. Normalized to 32MB:256KB cache configs.

tween those two, we simulate medium and high configurations to try to model cores with similar features to the ones we find in current server processors.

In terms of performance (see Figure 7a), for the majority of applications, low-end architectures are 35% slower than aggressive OoO configurations; 60% slower in the case of Specfem3D. Additionally, intermediate configurations suffer less than 5% slowdown in all applications except Specfem3D. We can appreciate similar results for 32 and 64 cores except in HYDRO and LULESH. As we mention in Section V-A applications with low parallelism leave many idle cores throughout the whole execution, therefore task distribution in 32 and 64 core configurations is the same.

The low-end pipeline configurations consume around 50% less power than its high-end counterparts (see Figure 7b), but as we discussed, these power savings come at a steep cost in terms of performance. On the other hand, intermediate *high* and *medium* OoO configurations consume 18% and 20% less power, respectively, across all applications, while still attaining performance that is close to the *aggressive* OoO pipelines. Therefore, the additional power that the aggressive cores consume is not translated into significant performance improvements, which makes the *high* and *medium* OoO configurations

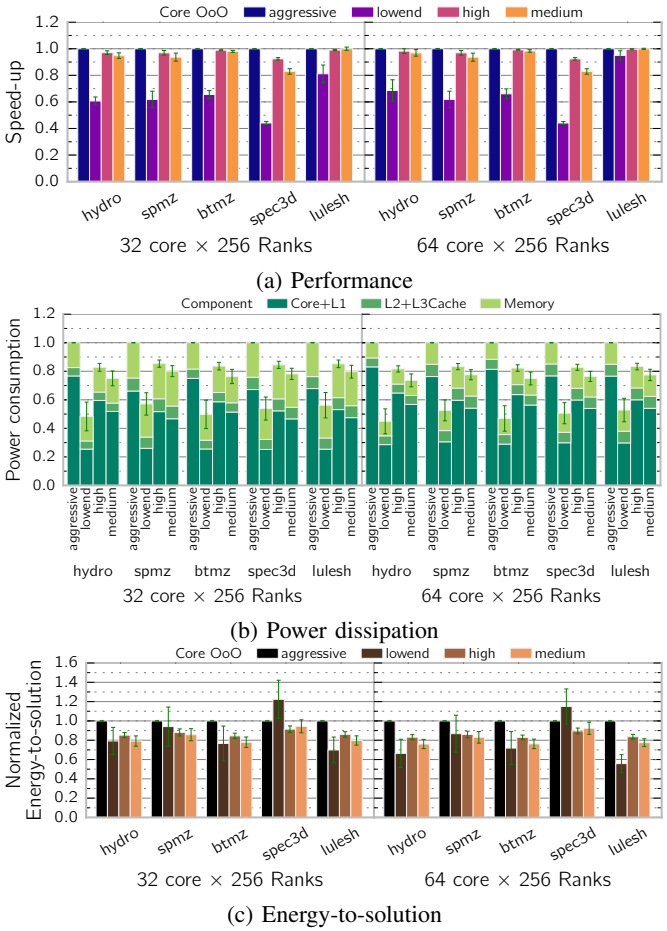


Fig. 7: Average simulation results varying core OoO structures. Normalized to aggressive configurations.

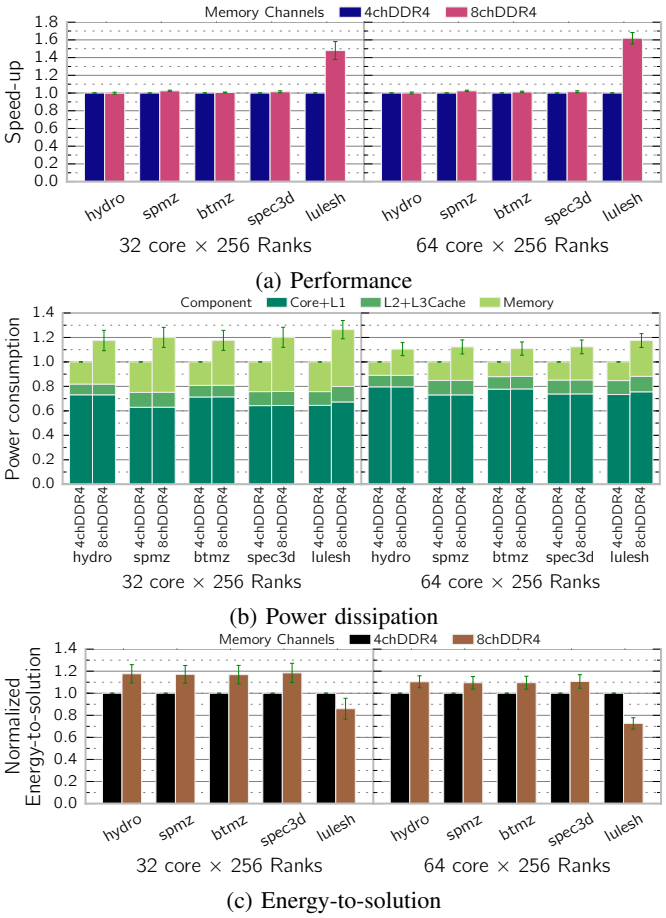


Fig. 8: Average simulation results increasing the number of memory channels. Normalized to four channel configurations.

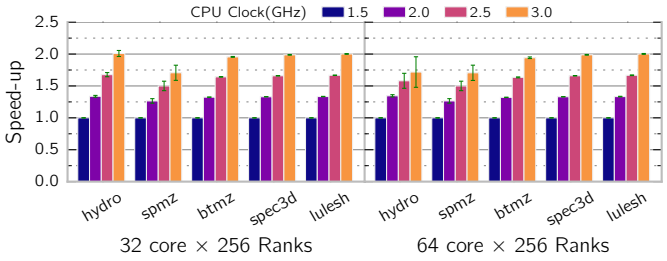
better design points for a good trade-off of performance and energy consumption. (see figure 7c). Additionally, we can observe that heavy memory constrained applications such as LULESH are able to obtain great energy savings since the impact in performance of the core compute capabilities is minor.

4) Memory channels: We evaluate memory configurations with four and eight DDR4 memory channels.

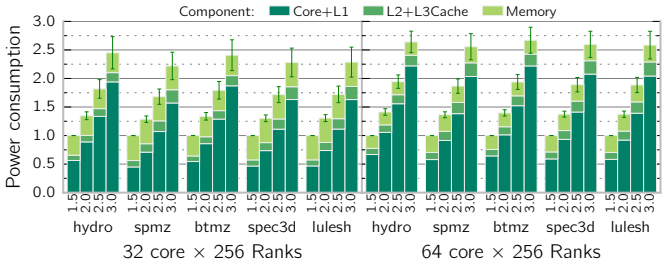
Figure 1 compares several memory metrics of our applications running on 64 core processors. Of all five, we find that only Specfem3D and LULESH have considerable high bandwidth requirements. Although Specfem3D requires more memory bandwidth than LULESH when simulating on a single core processor, at 64 cores it is the other way around: LULESH presents much better performance scaling and the usage of memory bandwidth scales accordingly. In Figure 8a we observe that despite the high bandwidth requirements of Specfem3D, increasing the number of memory channels from four to eight does not yield performance benefits. On the other hand, LULESH achieves up to 60% average speedup at 64 cores. Due to its very high memory bandwidth utilization, only LULESH takes advantage of having extra memory channels.

Upgrading from four to eight memory channels and populating them with DRAM DIMMs increases total DRAM power consumption by almost 100%. Nonetheless, in 64 core configurations, the impact of the memory component of this extra DIMMs over the overall total node power consumption is roughly 10%. As we see in Figure 8c LULESH, as a memory bound application, obtains on average 30% energy savings with eight channels.

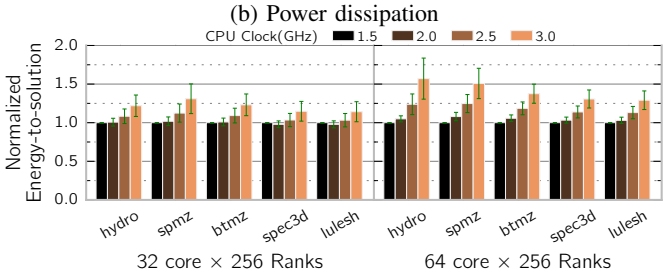
As a side note, memory bandwidth consumption is another computational characteristic that is expected to change significantly on improved versions of applications with better parallelized codes capable of scaling efficiently up to 64 cores. For example, if SPMZ was able to scale up to 64 cores with reasonable efficiency, it would demand more memory bandwidth than our four channel configurations are able to provide and we would obtain clear benefits on eight channel configurations. Another remark to consider is that, with the same micro-architectural configuration, the power consumed by memory can vary up to 20% from one application to another. If we compare HYDRO with applications with similar core occupancy such as LULESH, we observe that the total processor power consumption varies up to 5% in 64 cores configurations. Applications with many idle cores due to not having enough task level parallelism like SPMZ at 64 cores



(a) Performance



(b) Power dissipation



(c) Energy-to-solution

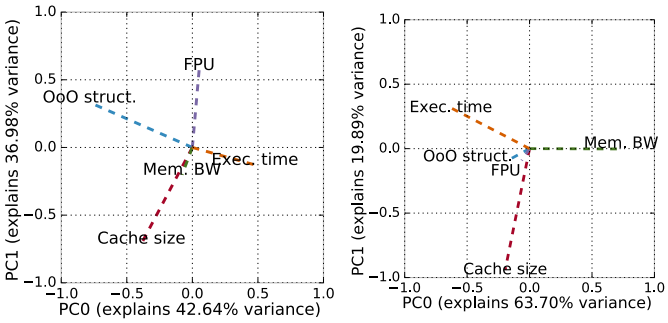
Fig. 9: Average simulation results increasing CPU Clock Frequency. Normalized to 1.5GHz configurations.

vary up to 20% difference in power consumption compared to HYDRO.

5) Frequency: We evaluate CPU frequency values from 1.5 to 3.0 GHz. For each step in frequency, we provide McPAT with adequate voltage parameters to scale up voltage accordingly to 22nm process technology. It is important to note that TaskSim uses the same frequency for all the components of the chip, so the frequency at which L1, L2, and L3 caches run is the same as the CPU clock.

Figure 9a shows that all applications except HYDRO scale their performance linearly as frequency increases. HYDRO encounters a scheduling bottleneck at high frequencies, tasks are too small and the threads are not able to schedule tasks fast enough. This is a TaskSim limitation as OpenMP/OmpSs runtime event timings are taken from the original trace. In real systems, this issue is not expected to happen.

Regarding power, Figure 9b shows that when comparing 1.5 to 3.0 GHz there is a 2× increase in performance and a 2.5× increase in power consumption. It is almost linear so we can consider that adding 1% in performance will increase by 1.25% the power consumption. Commonly known, scaling up frequency is a good way to obtain higher performance but it has a high power consumption cost. Frequency is a key aspect to consider and balance the different clock frequencies of the different hardware componentes in a compute node.



(a) HYDRO (b) LULESH

Fig. 10: PCA results

PCA results: performance correlation between different architectural paremeters.

C. Principal Component Analysis

In Section V-B, we provide average performance and standard deviation to measure the individual impact of each architectural feature. Even if bar plots are a convenient and straightforward way to visualize our experimental campaign, they do not provide insight about the performance tradeoffs between different architectural parameters. To explore those, we use Principal Component Analysis (PCA).

For each application, we find the principal components considering five variables: OoO capacity, number of memory channels, SIMD width, cache size and the number of cycles of that simulation.

Next, we discuss our results when applying PCA to HYDRO and LULESH. Other applications show similar trends and insights. We consider just the 2GHz CPU clock frequency and 64-core simulations.

Figure 10 shows the two most relevant principal components (PC) labeled as *PC0*, in the x-axis, and *PC1* in the y-axis. For the case of LULESH, PC0 explains more than 60% of the variance and it clearly shows that the memory bandwidth parameters evolve in a similar way as the total number of cycles. They evolve in an opposite way in the sense that an increase in memory bandwidth implies a reduction in the total amount of CPU cycles that LULESH requires. The cache size parameter has also a non-negative PC0 value, which means that it also related to the total cycles parameters for the case of LULESH, although not as much as the memory bandwidth. The other two considered parameters, OoO capacity and SIMD width, have no contribution to the PC0 variable for the case of LULESH, which means that the total cycles evolution is not related to them whatsoever.

In the case of HYDRO, the PC0 axis stands for 42.64% of the variance. Both the OoO capacity and the total number of cycles are major contributors to the PC0, which implies that they evolve in a tight and opposite way. The larger are the OoO capacities, the smaller becomes the total number of cycles for the case of HYDRO.

SPMZ specific config.				
Label	**Core OoO**	**FP Unit**	**Cache(L3:L2)**	**Memory**
DSE Best.	Aggressive	512-bit	96MB:1M	8-Ch. DDR4
Vector+	High ↓	1024-bit ↑	64MB:512kB ↓	4-Ch. DDR4 ↓
Vector++	High	2048-bit ↑	64MB:512kB	4-Ch. DDR4
LULESH specific config.				
Label	**Core OoO**	**FP Unit**	**Cache(L3:L2)**	**Memory**
DSE Best.	High	512-bit	96MB:1M	8-Ch. DDR4
MEM+	Medium ↓↓	64-bit ↓↓	64MB:512kB ↓	16-Ch. DDR4 ↑
MEM++	Medium	64-bit	64MB:512kB	16-Ch. HBM ↑

TABLE II: Application-specific architectural configurations. All use 64 core and 2GHz.

D. Unconventional Configurations

We test two additional pairs of unconventional configurations for the SPMZ and the LULESH codes (see Table II). We select these two applications as the results shown in Section V-B clearly show that SPMZ and LULESH are very sensitive to the SIMD register size and the memory bandwidth, respectively. In particular, results in Section V-B show how SPMZ benefits significantly from increasing the SIMD widths while other parameters such as the size of the OoO structures, cache and memory bandwidth have a minor impact in its performance.

Taking into account these observations, we simulate parallel executions of SPMZ considering architectures with increasing SIMD widths of 1024- (*Vector+* configuration) and 2048-bits (*Vector++* configuration) while keeping the rest of the architectural features settings that give the best possible performance-power tradeoff. LULESH is a heavily memory-bound application which does not benefit from floating point computing capacity. We test both high-bandwidth 16-Channel DDR4 (*MEM+*) and HBM (*MEM++*) configurations. For all results presented in this section, we compare the unconventional configurations against the best performing configuration in terms of execution time of Section V-B (*BEST-DSE*) running on 64 cores at 2GHz.

As we see in Figure 11, compared to *BEST-DSE*, the *Vector+* configuration for the case of SPMZ achieves a performance increase of $1.13\times$ with a similar increase in power while the more aggressive *Vector++* configuration obtains a performance benefit of $1.43\times$ but incurs in additional power consumption equivalent to $3.14\times$ the baseline. Overall, the *Vector++* suffers a pronounced $2.5\times$ increase in energy-to-solution.

With respect to LULESH, we achieve a 47% reduction in energy-to-solution by using narrower FPUs units and a performance increase of 7% by doubling up the memory bandwidth. Moreover, if we consider very low latency memory (*MEM++*), we can further achieve up to $1.30\times$ speedup over *DSE-BEST*. It is not possible to provide energy measurements regarding the *MEM++* due to the lack of data describing the energy consumption of the HBM technology, although previous studies indicate that it consumes less power than *MEM+* [31].

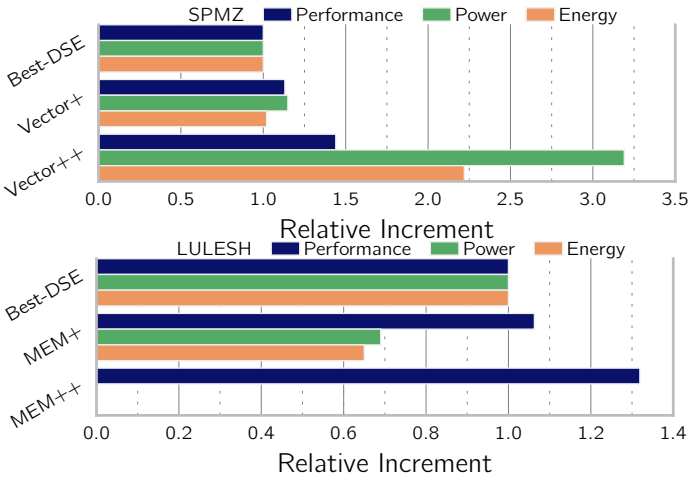


Fig. 11: Performance, power and energy-to-solution of application specific configurations. Normalized to the best result in Section V-B .

VI. Related work

Subsystem simulators are common tools that allow us to obtain performance prediction and assist computer architects into designing specific parts of the HPC systems. Mubarak et al. [32] propose CODES a fast and flexible simulation framework to model state of the art Torus and Dragonfly networks at a large-scale. Compared to this, our work focuses on a multi-level simulation of the whole parallel system.

Early proposals [33]–[35] offer solutions targeting large-scale systems capable of simulating thousands of nodes, but their frameworks focus mainly on network events and they not model CPU components or system software interactions in detail.

Wang et al. [36], present a multi-level simulation framework that is capable of modeling in detail many parts of the system architecture including power estimations but is limited to single node applications.

SST is a multi-scale simulator often used in combination with other simulators to model distributed applications. In BE-SST, authors combine SST with coarse-grained behavioral emulation models abstracting from microarchitectural details in favor of simulation speed. Other implementations integrate SST with a highly accurate simulator but require too costly full system simulations to produce a wide set of experiments [37], [38].

With the same objective, application specific analytical models [39], [40] use a small set of parameters to predict performance for a single application on large systems. Once those models are created and validated they are able to predict performance accurately with negligible compute and time cost. The main downside of these models is that they have little flexibility; any significant change in the application or hardware architecture requires the model to be updated, refined

and validated again. Our methodology focus on hardware microarchitectural exploration and iterative fast co-design; new features can be tested on all applications the moment they are included in the simulator with enough level of detail to study in depth hardware-software interactions.

VII. Conclusions

This paper contains a wide experimental campaign based on multi-level simulation methodologies. We extract several conclusions from this campaign which can be used to drive the design of next generation HPC systems. Our results show how compute node configurations with 512-bit wide FP units yield 20% to 75% performance speed-up and an average power increase of 60% with respect to 128-bit wide configurations. Consequently, it is appropriate to add 512-bit FP computing units to hardware devices, since these sizes may provide energy reductions for parallel codes that properly exploit both thread level and SIMD level parallelism. We also observe that in both 32- and 64-core processors, cache configurations with 1MB shared L3 cache and 512KB private L2 cache per core seem to offer the best trade-off in terms of power and performance. Moderate Out-of-Order capabilities (in terms of ROB size, Issue width, LD/ST buffers, FP Units) are a good design point in all of our tested applications. Memory bound codes benefit greatly (up to 60%) from enhanced memory bandwidth rates. Doubling the number of DDR channels (and populating them with DIMMs) increases the total node power consumption by only 10%-20%.

In addition to that, our scaling analysis of applications reveals that even without taking into account message passing communication overheads and using an optimistic upper bound (*hardware agnostic simulations*) four out of five applications are not capable of scaling over 75% parallel efficiency on 64 core CPU configurations. Further, we reproduced executions with up to 16,864 cores integrating all compute regions and MPI communications. In such case, parallel efficiency drops below 30%. With the continuous increase of static power expected in next-generation HPC systems, underutilization of compute resources is the main way to hurt overall energy efficiency. In view of the scaling results, we insist in the co-design point of view of bringing to the spotlight the necessity of a good parallel design of HPC applications that maximizes the utilization of the hardware resources available in each compute node.

References

[1] F. Haohuan, L. Junfeng, Y. Jinzhe, W. Lanning, H. Xiaomeng, Y. Chao, X. Wei, Q. Fangli, Z. Wei, Y. Xunqiang, H. Chaofeng, G. Wei, Z. Jian, W. Yangang, and Y. Guangwen, "The sunway taihulight supercomputer: system and applications," *SCIENCE CHINA Information Sciences*, vol. 59, no. 7, 2016.

[2] "Thunderx2 arm processors." [Online]. Available: cavium.com/product-thunderx2-arm-processors.html

[3] "SX-Aurora TSUBASA Architecture." [Online]. Available: nec.com/en/global/solutions/hpc/sx/architecture.html?

[4] "Fujitsu reveals details of processor that will power Post-K supercomputer." [Online]. Available: top500.org/news/fujitsu-reveals-details-of-processor-that-will-power-post-k-supercomputer/

[5] T. Grass, C. Allande, A. Armejach, A. Rico, E. Ayguadé, J. Labarta, M. Valero, M. Casas, and M. Moretó, "MUSA: a multi-level simulation approach for next-generation HPC machines," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016*, pp. 526–537.

[6] "GW4 Isambard." [Online]. Available: gw4.ac.uk/isambard/

[7] A. Sodani, "Knights landing: 2nd generation intel xeon phi processor," in *2015 IEEE Hot Chips 27 Symposium (HCS)*, 2015, pp. 1–24.

[8] A. Duran, E. Ayguad, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "OmpSs: A proposal for programming Heterogeneous Multi-Core Architectures," *Parallel Processing Letters*, vol. 21, no. 02, pp. 173–193, Jun. 2011.

[9] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *IEEE Computational Science and Engineering*, vol. 5, no. 1, pp. 46–55, Jan. 1998.

[10] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *SIGPLAN Not.*, vol. 40, no. 6, pp. 190–200, Jun. 2005.

[11] "DynamoRIO Dynamic Instrumentation Tool Platform." [Online]. Available: dynamorio.org/

[12] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A Fast and Extensible DRAM Simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, Jan. 2016.

[13] K. Chandrasekar, C. Weis, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, "Drampower: Open-source dram power & energy estimation tool," *URL: http://www.drampower.info*, vol. 22, 2012.

[14] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec. 2009, pp. 469–480.

[15] "Mont-blanc prototype: Dibona." [Online]. Available: montblanc-project.eu/prototypes

[16] "Sierra | High Performance Computing." [Online]. Available: hpc.llnl.gov/hardware/platforms/sierra

[17] "Marenostrum iv - user's guide." [Online]. Available: bsc.es/support/MareNostrum4-ug.pdf

[18] P.-F. Lavallée, G. C. de Verdiere, P. Wautelet, D. Lecas, and J.-M. Dupays, "Porting and optimizing hydro to new platforms and programming paradigms-lessons learnt," *Technical report, PRACE*, 2012.

[19] R. Teyssier, "Cosmological hydrodynamics with adaptive mesh refinement - A new high resolution code called RAMSES," *Astronomy & Astrophysics*, vol. 385, no. 1, pp. 337–364, Apr. 2002.

[20] R. F. H. vanderWijngaart, "NAS Parallel Benchmarks, Multi-Zone Versions," Phoenix, AZ, United States, Jun. 2003.

[21] I. Karlin, J. Keasler, and J. Neely, "Lulesh 2.0 updates and changes," Lawrence Livermore National Laboratory (LLNL), Livermore, CA, Tech. Rep., 2013.

[22] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *European Parallel Virtual Machine/Message Passing Interface Users Group Meeting*. Springer, 2004, pp. 97–104.

[23] "Extrae | BSC-Tools." [Online]. Available: tools.bsc.es/extrae

[24] A. Rico, F. Cabarcas, C. Villavieja, M. Pavlovic, A. Vega, Y. Etsion, A. Ramirez, and M. Valero, "On the simulation of large-scale architectures using multiple application abstraction levels," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 36:1–36:20, Jan. 2012.

[25] R. M. Badia, J. Labarta, J. Gimenez, and F. Escale, "Dimemas: Predicting mpi applications behavior in grid environments," in *Workshop on Grid Applications and Programming Tools (GGF8)*, vol. 86, 2003, pp. 52–62.

[26] "8gb (x72, ecc, sr) 288-pin ddr4 rdimm - micron technology." [Online]. Available: www.micron.com/~/media/documents/products/data-sheet/modules/rdimm/ddr4/asf18c1gx72pz.pdf

[27] S. Girona, J. Labarta, and R. M. Badia, "Validation of dimemas communication model for mpi collective operations," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer Berlin Heidelberg, 2000, pp. 39–46.

[28] S. L. Xi, H. Jacobson, P. Bose, G. Wei, and D. Brooks, "Quantifying sources of error in mcpat and potential impacts on architectural studies,"

in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015, pp. 577–589.

[29] "Fast and accurate dram power and energy estimation for dram." [Online]. Available: uni-kl.de/en/3d-dram/tools/drampower/

[30] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code," in *Proceedings of WoTUG-18: transputer and occam developments*, vol. 44, no. 1. IOS Press, 1995, pp. 17–31.

[31] J. Kim and Y. Kim, "Hbm: Memory solution for bandwidth-hungry processors," in *Hot Chips 26 Symposium (HCS), 2014 IEEE*. IEEE, 2014, pp. 1–24.

[32] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling Parallel Simulation of Large-Scale HPC Network Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 87–100, Jan. 2017.

[33] G. Zheng, G. Gupta, E. Bohm, I. Dooley, and L. V. Kale, "Simulating Large Scale Parallel Applications Using Statistical Models for Sequential Execution Blocks," in *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, Dec. 2010, pp. 221–228.

[34] E. Grobelny, D. Bueno, I. Troxel, A. D. George, and J. S. Vetter, "FASE: A Framework for Scalable Performance Prediction of HPC Systems and Applications," *SIMULATION*, vol. 83, no. 10, pp. 721–745, Oct. 2007.

[35] W. E. Denzel, J. Li, P. Walker, and Y. Jin, "A Framework for End-to-End Simulation of High-performance Computing Systems," *SIMULATION*, vol. 86, no. 5-6, pp. 331–350, May 2010.

[36] J. Wang, J. Beu, R. Bheda, T. Conte, Z. Dong, C. Kersey, M. Rasquinha, G. Riley, W. Song, H. Xiao, P. Xu, and S. Yalamanchili, "Manifold: A parallel simulation framework for multicore systems," in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2014, pp. 106–115.

[37] M. Hsieh, K. Pedretti, J. Meng, A. Coskun, M. Levenhagen, and A. Rodrigues, "Sst + gem5 = a scalable simulation infrastructure for high performance computing," in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, ser. SIMUTOOLS '12, 2012, pp. 196–201.

[38] A. Ramaswamy, N. Kumar, A. Neelakantan, H. Lam, and G. Stitt, "Scalable behavioral emulation of extreme-scale systems using structural simulation toolkit," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. ACM, pp. 17:1–17:11.

[39] V. Marjanović, J. Gracia, and C. W. Glass, "Performance modeling of the hpcg benchmark," in *International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*. Springer, 2014, pp. 172–192.

[40] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, "Predictive performance and scalability modeling of a large-scale application," in *Supercomputing, ACM/IEEE 2001 Conference*. IEEE, 2001, pp. 39–39.